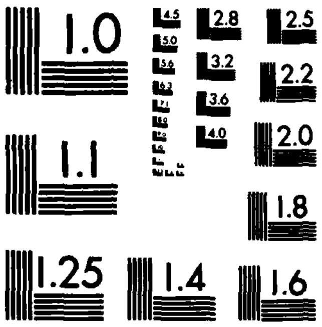
AD-A154 377 A PARALLELIZED POINT SUCCESSIVE OVER-RELAXATION METHOD ON A MULTIPLE INST..(U) ARMY BALLISTIC RESEARCH LAB ABERDEEN PROVING GROUND MD N R PATEL ET AL. NOV 84 1/1

UNCLASSIFIED BRL-MR-3411 SBI-AD-F300 539 F/G 9/2 NL

END

FILMED

DTIC


1.0
4.5
5.0
5.6
6.3
7.1
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

(2)

BRL

MEMORANDUM REPORT BRL-MR-3411

A PARALLELIZED POINT SUCCESSIVE OVER-RELAXATION METHOD ON A MULTIPLE INSTRUCTION MULTIPLE DATA STREAM COMPUTER

Nisheeth R. Patel
Walter B. Sturek
Harry F. Jordan

November 1984

US ARMY BALLISTIC RESEARCH LABORATORY
ABERDEEN PROVING GROUND, MARYLAND

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER MEMORANDUM REPORT BRL-MR-3411	2. GOVT ACCESSION NO. AD-A154377	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A PARALLELIZED POINT SUCCESSIVE OVER-RELAXATION METHOD ON A MULTIPLE INSTRUCTION MULTIPLE DATA STREAM COMPUTER		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Nisheeth R. Patel* Walter B. Sturek Harry F. Jordan**		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS U.S. Army Ballistic Research Laboratory ATTN: AMXBR-LFD Aberdeen Proving Ground, Maryland 21005-5066		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS RDT&E 1L161102AH43
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Ballistic Research Laboratory ATTN: AMXBR-OD-ST Aberdeen Proving Ground, Maryland 21005-5066		12. REPORT DATE November 1984
		13. NUMBER OF PAGES 39
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release, distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES This report supersedes IMR No. 811, dated April 1984. **University of Colorado Electrical and Computer Engineering Department Boulder, CO 80309 *Under contract to National Research Council		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Parallelized Point Relaxation Method; Parallel Processing; Rowwise Natural Ordering; Successive Over-Relaxation Method; MIMD; Multiprocessor		

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A parallelized point rowwise Successive Over-Relaxation (SOR) iterative algorithm is developed for the Heterogeneous Element Processor (HEP) multiple instruction stream computer. The classical point SOR method is not easily vectorizable with rowwise ordering of the grid points, but it can be effectively parallelized on a multiple instruction stream machine without suffering in convergence rate. The details of the implementation including restructuring of a serial FORTRAN program and techniques needed to exploit the parallel

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT (Continued)

processing architectural concept of the HEP are presented. The parallelized algorithm is analyzed in detail. The lessons learned in this study are documented to provide guidelines for similar future coding since new approaches and restructuring techniques are required for programming a multiple instruction machine which are totally different than those required for programming an algorithm on a vector processor. To assess the capabilities of the parallelized algorithm, it was used to solve Laplace's equation on a rectangular field with Dirichlet boundary conditions. Computer run times are presented which indicate significant speed gain over a scalar version of the code. For a moderate to large size problem, seventeen or more processes are required to make efficient use of the parallel processing hardware. Also, to demonstrate the capability of the algorithm for a realistic problem, a numerical solution was obtained for a viscous incompressible fluid in a square cavity. Since point iterative relaxation schemes are at the core of many systems of elliptic as well as non-elliptic partial differential equations occurring in engineering and scientific applications, the present study suggests the possibility of both reducing the real time processing and increasing the scope of computational modeling.

TABLE OF CONTENTS

		Page
	LIST OF ILLUSTRATIONS	5
I.	INTRODUCTION	7
II.	THE HEP AND HEP FORTRAN	8
III.	ITERATIVE RELAXATION METHODS	9
IV.	IMPLEMENTATION	11
V.	RESULTS	15
VI.	APPLICATIONS	18
VII.	CONCLUSIONS	22
	ACKNOWLEDGEMENTS	22
	REFERENCES	34
	DISTRIBUTION LIST	37

LIST OF ILLUSTRATIONS

Figure		Page
1	SIMD Versus MIMD Pipelining	24
2	Uniform Mesh System for the Model Problem	25
3	Speed of First FORTRAN Version	26
4	Speed of Optimized FORTRAN Version	27
5	Speed of Version with Assembly Language Kernel	28
6	Effects of the Critical Section for Many Processes	29
7	Critical Section Removed from the Inner Loop	30
8	Speed of Four PEM Version with Assembly Language j Line Sweep	31
9	Speed for Four PEM FORTRAN Version	32
10	Streamline Patterns and Stream Function Values for Flow in a Square Cavity, Re = 100	33

I. INTRODUCTION

In the last decade, significant progress has been made in developing better algorithms that are used for numerical solution of partial differential equations. Computer codes employed in many engineering applications require massive calculations and still use large amounts of computer resources. A basic requirement is that the algorithm be efficient. In practical terms this means obtaining the solution with desired accuracy using the least amount of computer resources. A detailed discussion of improvement in various numerical algorithms transcends the material in this report. The available computer architecture, whether it be of the serial, vector or parallel type, tends to influence the development of these algorithms. The advent of high performance sixth generation computers, such as the CYBER-200 and CRAY-1 series, provides an important breakthrough for computationally demanding engineering problems.[1] These supercomputers incorporate vector processing capabilities to provide the computational power required by large scale numerical simulation. The above-described vector processing computers employ pipeline processors. Pipelining is a technique of decomposing a sequential operation into suboperations with each suboperation being executed in a special dedicated segment that operates concurrently with all other segments. The name "pipeline" implies a flow of information analogous to an industry assembly line. It is characteristic of pipelines that several operations progress in distinct segments at the same time.

Vector processors are called Single Instruction Multiple Data (SIMD) computers because one instruction causes operation on all components of one, or a pair of vectors. Currently the most cost effective way to implement an SIMD computer is to use pipelining. In pipelined SIMD machines the operating units are broken into small stages with data storage in each stage. Complete processing of a pair of operands involves the data passing sequentially through all stages of the pipeline. Different pairs are essentially elements of vectors being operated upon.

The Heterogenenous Element Processor (HEP), which is a parallel processor, uses pipelining to implement multiple processes. In this computer Multiple Instruction Streams act on Multiple Data items in parallel (MIMD). The HEP computer consists of one or more pipelined MIMD Process Execution Modules (PEMs). The pipelining in the HEP can be summarized as follows. Independent instructions, including operands, flow through the pipeline with an instruction being completely executed in eight steps. Independence of activity in successive stages of the pipeline is achieved not by processing independent components of vectors but by alternating instructions from independent instruction streams in the pipeline. Figure 1 shows differences

1. *R. D. Levin, "Supercomputers," Scientific American, January 1982, p. 118.*

between pipelined SIMD and pipelined MIMD. A detailed discussion of thearchitecture and application programs on the HEP can be found in Reference 2.

The purpose of this report is to document the development of a preliminary parallelized algorithm for an iterative solution method which has proven to be reliable and competitive for classes of linear and nonlinear, elliptic and non-elliptic partial differential equations arising from a broad range of scientific applications. For complex problems, the iterative solution method to be explored on the HEP is not easily vectorizable on vector processors in its original form, but it can be efficiently parallelized on the HEP. Since this iterative scheme is at the core of many large scale scientific codes and generally consumes most of the computer time, the performance of the parallelized algorithm will be evaluated to give some idea of possible speed-up on this parallel system. Also, lessons learned in programing the code on the HEP will be documented and will provide guidelines for similar future coding.

II. THE HEP AND HEP FORTRAN

In this section, some extensions of standard FORTRAN and some intrinsic functions will be described since they are an important part of the development of the parallelized algorithm presented in this report. Details of intrinsic functions and the HEP FORTRAN compiler can be found in Reference 3. There are four types of memory in a HEP system: Program memory, register memory, constant memory, and data memory. Program, register, and constant memories are local to a PEM. Data memory is shared between PEMs. On a single PEM, multiple processes (each process being a separate instruction stream) will operate for a given job or task and share memory. Out of 128 possible processes available on a PEM, 64 are for supervisors and 64 are for users. Taking into account the overhead involved, it is generally safe to use a maximum of 50 user processes.

The pipelined nature of the HEP is not of particular significance to the programmer's view of the system. The general characteristics of the HEP are that it is a shared-memory, multiple instruction stream computer with enough hardware support for process scheduling to allow up to three times as many processes to be active as the actual amount of parallel hardware without suffering any scheduling penalty. The number of instruction streams which can be processed in parallel is a moderately complex function of the pipeline structure but is about 10-15 per PEM for an average instruction mix. The user can think of writing a collection of separate FORTRAN programs which will execute together to do a calculation. In many cases a single program or subprogram may be written and copies of it executed in parallel by several user processes. It is possible to create several separate instruction streams or processes using a CREATE statement. The CREATE statement can be thought of

2. *H. F. Jordan, "Experience with Pipelined Multiple Instruction Streams," Proc. IEEE, Vol. 72, No. 1, January 1984, pp. 113-213.*

3. *HEP FORTRAN 77 USER'S GUIDE, Reference Manual, Denelcor Inc., Denver, Colorado, February 1982.*

as the CALL statement required for a subroutine. However, control returns immediately from a CREATE statement but the created subroutine, with a unique copy of its local variables, is also executing simultaneously. The RETURN statement terminates the process executing in a created subroutine.

For correct interaction among parallel operating processes, each cell in register and data memory is assigned a full/empty state. Asynchronous variables have access to the full/empty state and are used for synchronization. Asynchronous variables are identified by names beginning with a $ symbol and may be of any type allowed by FORTRAN. They also adhere to implicit typing rules of FORTRAN based on the letter which follows the $ sign. The following properties pertaining to asynchronous variables are used for synchronizing processes. When an asynchronous variable appears on the right hand side of an assignment statement, it must wait for full, read and set empty, and when it appears on the left hand side of the assignment statement it must wait for empty, write and set full. A PURGE statement will set the state (not the value) of the asynchronous variables to empty, regardless of their previous state. The intrinsic function VALUE reads the value of an asynchronous variable regardless of its state and does not change the state. The function SETE ignores previous state and sets the state to empty. The function WAITF waits for an asynchronous variable to be full, reads the value but does not set the state to empty. This completes a brief review of some important features of the HEP and HEP FORTRAN.

III. ITERATIVE RELAXATION METHODS

The point iterative relaxation techniques are at the heart of many systems of elliptic and non-elliptic partial differential equations occurring in fluid dynamics, heat transfer, nuclear and plasma physics, electric networks, semi-conductor device modeling, meteorology, and structural analysis. Due to the large number of equations involved, usually direct methods are not suitable in terms of storage and efficiency. For comparative study, let us consider some iterative relaxation techniques[4] which cover large classes of problems. The computer time required for large scientific problems is so large that any increase in efficiency can represent substantial savings. As iterative relaxation methods represent the most time consuming part of many large codes, we will consider a simple model problem to analyze and investigate the possibilities of increasing the computational efficiency by developing algorithms which fit the architecture of the computer available. Note that a square, or even rectangular boundary, is not required by the MIMD algorithm to be described. It is only required that boundary values coincide with mesh points. For illustrative purposes, consider the simple model problem: Laplace's equation in two-dimensions with simple Dirichlet boundary conditions.

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0 \qquad (1)$$

4. *R. S. Varga, Matrix Iterative Analysis, Prentice-Hall, Inc., New York, New Jersey, 1962.*

The field is subdivided into square cells of $\Delta x = \Delta y = h$ as shown in Figure 2. Using central finite-difference approximations at a mesh point (X_i, Y_j) Equation (1) can be written as:

$$4f_{i,j} - f_{i+1,j} - f_{i-1,j} - f_{i,j+1} - f_{i,j-1} = 0 \qquad (2)$$

The boundary conditions imposed are +1.0 on the left and bottom boundary and -1.0 on the right and top boundaries.

Equation (2) can be solved iteratively for field unknowns using:

$$f_{i,j}^{(n+1)} = 1/4\,(f_{i+1,j}^{(n)} + f_{i-1,j}^{(n)} + f_{i,j-1}^{(n)} + f_{i,j+1}^{(n)}) \qquad (3)$$

This is the point Jacobi method. For this particular problem the method simply involves setting the new iterate equal to the arithmetic average of its four neighbors. Superscript n denotes iteration level. Note that on the right hand side we require all information from the previous iteration level. Thus the method is highly parallelizable since we can solve for all field iterates simultaneously. However, it gives a slower rate of convergence compared to the point Successive Over-Relaxation (SOR) method in many cases. The point SOR method applied to the field unknowns updated in the natural rowwise order can be represented by the following equation.

$$f_{i,j}^{(n+1)} = \frac{w}{4}\,(f_{i+1,j}^{(n)} + f_{i-1,j}^{(n+1)} + f_{i,j+1}^{(n)} + f_{i,j-1}^{(n+1)}) + (1 - w)\,f_{i,j}^{(n)} \qquad (4)$$

In Equation (4), w is a relaxation parameter used to accelerate convergence and superscript n denotes iteration level. With $w = 1$ the method reduces to the Gauss-Seidel iterative method. Also, for $w < 1$ it gives under relaxation and for $w > 1$ gives over relaxation. It can be shown for many problems that with a suitably chosen relaxation parameter, the point rowwise SOR method gives substantially larger asymptotic rates of convergence than those for the point Jacobi and point Gauss-Seidel iterative methods. The point SOR method described above uses advanced (n+1) values at two neighboring points, (i-1,j) and (i,j-1). The point Jacobi method in Equation (3) is a two level equation requiring storage of f^{n+1} and f^n. The point SOR requires storage for only one set of f values, which are continually overwritten until convergence is attained. For the Dirichlet boundary conditions, if any term on the right-hand side of Equation (4) belongs to the boundary, the corresponding term is understood to have the prescribed boundary value. The point rowwise SOR method of Equation (4) will be shortened to the SOR method in what follows.

IV. IMPLEMENTATION

Because of its effectiveness and simplicity, the SOR method, which is frequently used, is reliable and very competitive for many problems. A FORTRAN program for the model problem on a serial computer would include the following statements.

```
          OMW = 1. - W
          W04 = W * 0.25
          DO 16 J=2, JL-1
          DO 18 I=2, IL-1
          TF = W04 * (F(I+1,J) + F(I-1,J) + F(I,J+1) + F(I,J-1))
                    + OMW * F(I,J)
          ERR = ABS(TF - F(I,J))
          ERRMX = AMAX1(ERRMX,ERR)
          F(I,J) = TF
     18   CONTINUE
     16   CONTINUE
              "
              "
```

A simple procedure is used for computing the error norm, and scalar ERRMX is used for finding the maximum error norm in the field. ERRMX is compared to the allowable error norm for checking the convergence of the solution procedure. As shown above, the SOR method can be easily implemented on a serial machine. For complex problems, the equations used are usually more involved, but the basic features remain essentially the same.

Now let us consider the implementation of the SOR method on vector processing computers. As mentioned before, the convergence rate of the SOR method depends partly upon using updated values at adjacent points while solving for a given point. Also, for vector processors, vectors must be stored and must be available to make use of concurrency in the computer. Thus for complex problems involving cross derivatives, the SOR method is not easy to vectorize in its original form. However, there exists a simple system ordering for solution of partial differential equations using a rectangular grid which will vectorize easily. Sufficiently large vectors can be formed by associating vectors with alternate field points (red-black checkerboard pattern). This modified SOR is referred to as checkerboard SOR and can be effectively implemented on vector processors. Since the evaluation of cross derivatives is lagging one stage per iteration, the convergence rate of the checkerboard SOR may not be as good as the SOR method, and schemes involving more than two colors to decouple the grid have been considered by Adams and Ortega.[5] The comparison of various vectorized relaxation schemes for some specific fluid

5. *L. M. Adams and J. M. Ortega, "A Multi-Color SOR Method for Parallel Computation," Proc. 1982 Intl. Conf. on Parallel Processing, Bellaire, MI, August 1982, pp. 53-58.*

flow problems is reported in Reference 6. It was found that the convergence rate of checkerboard SOR was not as good as that of the successive line over-relaxation scheme. Also, for complex problems there exist mathematical analyses for point SOR for finding a nearly optimum local acceleration parameter; however, for checkerboard SOR there is no such theory. The detailed information about the checkerboard SOR implementation on the CYBER-200 series vector computers for implicit finite-difference solution of incompressible Navier-Stokes equations can be found in Reference 7. Implementation of checkerboard SOR on a vector processor is somewhat involved. The implicit vectorization option of the compiler is not of much use when implementing the checkerboard SOR. The sequential machine SOR method will not produce any significant improvement in speed using the implicit vectorization option on a vector processor. The implementation on the vector processor requires development of a system dependent algorithm and code. The procedure for handling irregular domains on a vector processor, even though the boundary coincides with Cartesian grid nodes and no interpolation is required for implementing boundary conditions, becomes less efficient due to the increase in the bookkeeping overhead. Frequently, the use of arbitrary coordinate transformation introduced by boundary conforming coordinate systems is desired. The use of boundary conforming coordinate systems, in which coordinates match the boundary, allow all computations to be done on a fixed rectangular grid and gives a well ordered system of algebraic equations which can efficiently use vector processors.

For the Laplace's equation, which is used as a model problem in this study, the convergence rate of the checkerboard SOR and the SOR method is the same. Although the checkerboard SOR is highly parallelizable, the major thrust of this study was to develop a parallelized point iterative relaxation algorithm for fluid flow equations involving cross derivatives. Therefore, the SOR method with natural rowwise ordering is considered. Now let us explore the possibility of implementing the SOR method on the HEP multiprocessor. It has been found that the SOR method can exploit the architecture of the HEP as effectively as it can that of serial machines; and that it can also increase computation speed by an order of magnitude over serial machines due to parallel processing capabilities. In the remainder of this section, we will concentrate on the implementation of the SOR method on the HEP. Some features of the present approach for parallelized SOR methods are as follows. Each row (i.e., j line) is swept in sequential manner using a DO LOOP, which guarantees that computation for a given j line at node i will not start until the computation at node i-1 is complete. This is essentially the DO LOOP 18 of the serial machine program given above. The computations on j lines are carried out in parallel. Let us say we use 5 parallel processes for this particular case. One of the requirements for the SOR method is that we must use advanced values at node i,j-1 while solving for a node i,j. This means that

6. *P. F. Bradley, D. L. Dwoyer and J. C. South, "Vectorized Schemes for Conical Flow Using the Artificial Density Method," AIAA Paper Number 84-0162, January 1984.*

7. *N. Patel and J. F. Thompson, "A Vectorized Solution for Incompressible Flow," AIAA Paper No. 84-1534, June 1984.*

computations at line j = 3 for i = 5 must not start until computations at line j = 2 for i = 5 are complete, and similarly for line j = 4 and so forth. In this case, with the number of parallel processes equal to the number of j field lines, the computational wave can be denoted by a diagonal line. The FULL/EMPTY property of asynchronous variables is very useful in implementing this requirement and guarantees the required synchronization among all processes operating in parallel. If all values in the interior of the array start off empty but the j = 1 boundary values are full, then the synchronization is done as follows. Of the five array values read, only the state of the (i,j-1) element is observed by the process computing line j. Waiting for this to be full guarantees that the new value is obtained for the (i,j-1) element stored by the process computing line j-1. In turn, the process computing line j stores the new (i,j) element and sets it full for use by the process computing line j+1. All other values are read without using or changing the full/empty state. An old (i+1,j) value and a new (i-1,j) value are guaranteed by the sequentiality of the process computing line j. The old value at (i,j+1) is insured because the process computing line j+1 cannot store the new (i,j+1) value until after it has read the (i,j) value being computed by this process. In general, each parallel process performs a rowwise relaxation sweep, and asynchronous variables are used to produce the proper lag between parallel processes operating on adjacent rows. As soon as the first row sweep of parallel process #1 is complete, it will pick up the index value of the next unswept row, waiting in line in case the number of j field lines is greater than the number of parallel processes being used.

A sample HEP FORTRAN program is shown below to more fully illustrate the procedure.

```
        DIMENSION $F(102,102)
        COMMON/BLK1/    $F,$JJ,ILM1,JLM1,....
        COMMON/BLK2/    $ERNM
             "
             "
        IL = 102
        JL = 102
        ILM1 = IL - 1
        JLM1 = JL - 1
        IPRC = 5            (Number of parallel processes)
        IPRCM1 = IPRC - 1
             "
             "
 15     CONTINUE            (Iteration loop)
             "
             "
        PURGE $JJ,$NNN
        $JJ = 2
        $ERNM = 0.0
             "
             "
        DO 1313 NN=1, IPRCM1        (Create parallel processes)
        $NNN = NN
        CREATE SOR ($NNN)
 1313   CONTINUE
```

```
          $NNN = IPRC
          CALL SOR ($NNN)
          IF (ERRNRM - RESMAX) 15,23,23
   23     CONTINUE
          STOP
          END


          SUBROUTINE SOR ($NNN)
          DIMENSION $F(102,102)
          COMMON/BLK1/   $F,$JJ,ILM1,JLM1,...
          COMMON/BLK2/   $ERNM
            "
            "
          NN = $NNN
   11     J = $JJ          (make a local copy of row number, i.e., J line)
          $JJ = J + 1
          IF (J .GT. JLM1) GO TO 33
            "
            "
          W04 = W * 0.25
          OMW = 1.0 - W
            "
            "
          DO 22 I=2, ILM1
          TF = W04 * (VALUE ($F(I+1,J)) + VALUE ($F(I-1,J))
                     + VALUE ($F(I,J+1)) + $F(I,J-1))
                     + OMW * VALUE ( $F(I,J))
          $ERNM = AMAX1 (ABS (TF - VALUE ($F(I,J)), $ERNM)
          $F(I,J) = TF
   22     CONTINUE
          GO TO 11
            "
          RETURN
          END
```

Some comments on the right hand side are used for explanation of the code. Note that we have to initialize some asynchronous variables before we create five parallel processes. DO LOOP 1313 creates four processes and the fifth parallel process is invoked using a CALL statement. Note that the loop uses asynchronous variable $NNN which essentially passes the process number. The use of $NNN insures that no process will be created until the previously referenced process has obtained its process number. This approach is necessary because, due to overhead involved in creating a process, the value of a loop index might have changed by the time the created process references the index via its argument. The subroutine SOR is written for a self scheduling partition of the work. In self scheduling, each process uses a new value of the asynchronous index $JJ, increments it, and refills it using a local variable J. Thus a unique J line is obtained and the process proceeds with the local value of J. The process terminates when the value of $JJ becomes greater than JL-1. Note that $JJ was initialized to 2. DO LOOP 22 is a row-wise sweep and solves the governing equations. In this program five processes are running in parallel. Variable \$F is shared by more than one process through a common block. Due to the basic requirement of the algorithm, to use updated values at (i-1,j) and (i,j-1), the FORTRAN statement does not use the VALUE function for \$F(i,j-1). Due to the full/empty property of asynchronous variables, a process waits until a new value for \$F(i,j-1) becomes available, then consumes it, computes a new value of \$F(i,j), and sets it full to make it available to the processes computing line j+1.

For computing an error norm, in this case, an asynchronous variable \$ERNM is used. Also it is required that no progress beyond the CALL statement can be made until all processes are completed. This can be done by using asynchronous variables and a counter to count the number of completed processes. Note that the scheme is easily adapted to irregular boundaries having the Cartesian grid nodes on the boundary (unequal j lines). It is only desirable not to have the j lines too short on the average. This can be done by orienting the region properly.

The above described computer program for the parallelized SOR method is rather imprecise in the sense that it is not a copy of a working program. A number of other issues such as boundary conditions, initialization, etc., must be addressed before it can be made into a working program. However, it does give a clear picture of how the SOR method can be parallelized on the HEP. The above described programming example shows one possible way of implementing the algorithm on the HEP. The same algorithm can be programmed using some other parallel programming options and techniques, depending on ease in programming, personal choice and computer efficiency.

V. RESULTS

Some interesting results of the present study are summarized in this section. The discussion of the results is focused on the computational speed-up that can be achieved using a parallelized code and some techniques which help make code efficient. In all cases 102 x 102 grid points were considered. Excluding boundary nodes this gave a field size of 100 x 100. The scheme of using different processes to sweep j-lines means that there are 100 pieces of work to be divided among multiple processes on each iteration.

To evaluate performance of the parallelized SOR method on the HEP, a similar code for a sequential machine was developed. A sequential version of the SOR method was run on the VAX-11/780 computer with the same boundary conditions and the same number of grid points. An error norm was defined as the maximum change in solution between two successive iteration levels. The error norm was compared to an acceptable tolerance. When the value of the computed maximum error norm was equal to or less than 0.0001, the solution procedure was stopped. The above convergence criteria was used for all computations. Since the convergence rate of the SOR method is very sensitive to the value of acceleration parameter used, some numerical experiments were carried out on the VAX-11/780 for finding a nearly optimum constant acceleration parameter for a given problem. Computer runs were made for acceleration parameter values from 0.1 to 1.9. It was found that the value of the nearly optimum constant acceleration parameter which required a minimum number of iterations for a given error tolerance was about 1.5. The value of the acceleration parameter used for all runs on the HEP was 1.5. For the model problem, the execution time on the VAX-11/780 was about 479 seconds, using single precision (32 bit) option. It must be noted that all computer runs on the HEP were made using 64 bits. Also, the parallelized code was run on the HEP using a single PEM unless otherwise noted.

The first version of the parallelized SOR HEP code, which was rather crude, gave best execution time of about 52.7 seconds when using 20 parallel processes. The variation in execution time versus the number of active parallel processes is shown in Figure 3. Note that there is a sharp drop in execution time for 1 to 10 procceses, and then the curve becomes flat as the increase in number of processes fills the execution pipeline. Next the original code was slightly modified by looking at the number of computer instructions required for a given FORTRAN statement. Since there is no compiler optimization option available at this stage, an attempt was made to manually optimize the code by forcing some local variables to be held in registers using a compiler extension. There are about 40 registers available to each process. The use of registers instead of data memory saves some data memory instructions and increases computational efficiency. Figure 4 shows the graph of execution time versus active parallel processes for this version of the code. The execution time came down to 30.4 seconds for 20 parallel processes. When the optimization option becomes available, these modifications will no longer be user dependent and the compiler can take care of them. Further optimization can be achieved by writing the inner loop (sweep across a j line) in assembly language. This was done and gave the results shown in Figure 5. The factor of 3.7 speedup from Figure 4 to Figure 5 is partially a result of a slight algorithm improvement; but the factor of about three is due entirely to the non-optimizing compiler.

The graph of Figure 4 is shown in an expanded scale in Figure 6. Note there is a sharp rise in execution time for greater than 25 processes. This is a result of the simple treatment of the error norm computation. The asynchronous accesses to $ERNM ensure that only one process may update it at a time. Even though only three machine instructions are required for this update, the execution time of the loop rapidly becomes dominated by the need for processes to pass, one at a time, through these three instructions as the number of processes increases. This problem of so-called critical sections of code which can be executed by only one process at a time is well known in the field of parallel computing. The curve exhibits the form predicted by a simple critical section model. Assume an amount T of work which can be spread over an arbitrary number P of processes with the only parallelization cost being a critical section of size T_c. The total work in the parallelized code is $T_p = T + P \times T_c$, while the amount done by a single process is $t = \frac{T}{P} + T_c$. If work is measured in units of execution time, the best case time for execution with P processes is $T_b = \frac{T}{P} + T_c$. This can only happen when $(P - 1) \times T_c < \frac{T}{P}$ and it happens that a process reaches its critical section only while others are doing useful work. When $(P - 1) \times T_c > \frac{T}{P}$ the time is dominated by that needed for P processes to pass sequentially through their critical sections. Thus $T_b = T_c + \max\left(\frac{T}{P}, (P - 1) \times T_c\right)$.

The justification for using the best case analysis is that if the parallelized work is executed repetitively, as in a loop, the initial delays will bring the processes into best case synchronism on subsequent executions. The maximum speedup is seen to occur for P_c processes where $P_c \times (P_c - 1) \times T_c = T$ or, for $\frac{T}{T_c} \gg 1$, $P_c \approx \sqrt{\frac{T}{T_c}}$.

Incorporating a limitation U on the amount of parallelism actually available in the hardware, the execution time model becomes:

$$T_b = T_c + \max\left(\frac{T}{\min(P,U)}, (P - 1) \times T_c\right).$$

The data of Figures 4 and 6 would suggest a hardware limit of about $U = 7.5$ and a $\frac{T}{T_c} \approx 375$. U is estimated from the maximum speedup and the ratio from the slope of the rising tail. The measured U is somewhat smaller than would be expected from the HEP hardware because part of the hardware parallelism is used to speed up a single process through instruction lookahead.

The solution is to let each process compute a local error norm for the elements of F which it has computed and combine this local norm into a global one as the process terminates. The code becomes:

```
              "
              "
        DO 22 I=2, ILM1
              "
              "
        LOCERM = AMAX1 (ABS(TF - VALUE(F(I,J))), LOCERM)
        $F(I,J) = TF
 22     CONTINUE
        GO TO 11
 33     CONTINUE
        $ERNM = AMAX1 (LOCERM,$ERNM)
              "
              "
        RETURN
        END
```

This moves the critical section outside both loops and gives the results shown in Figure 7. The best execution time was reduced to 29 seconds and the rising tail for large numbers of processes was completely eliminated.

The discontinuities in Figure 7 can be explained in the following way. We have 100 J lines. Therefore, when using 20 processes, the task can be completed in 5 stages. When the number of processes divides 100 evenly, good execution timing occurs. However, when the number of parallel processes is equal to 24, (which does not divide 100 evenly), 96 J lines are completed in the first 4 stages. In the last stage, only 4 processes make use of the computer and execution time goes up. Since process creation overhead is large and this is done only once, by giving enough work to each process the overhead can be made negligible relative to total execution time. Thus the parallelized approach is cost effective for moderate to large size problems.

To obtain greater speed, the parallel SOR program was executed on all four PEMs of the available HEP system. The single PEM algorithm with j line sweep in assembly language was extended to four PEMs with timing results shown in Figure 8. The fastest execution time was 4.5 seconds with 51 processes. For comparison, the single PEM gave a fastest execution time of 14.1 seconds with 17 processes. Also a version written entirely in FORTRAN was executed on four PEMs with the results shown in Figure 9. The fastest execution time was 6.9 seconds with 76 processes compared to 29.0 seconds with 17 processes for the single PEM FORTRAN version. Finally, to get an upper performance limit, the full SOR iteration was coded in assembly language and achieved the fastest execution time of 3.0 seconds with 76 processes. An often used (and misused) figure of merit for a computer is millions of floating point operations per second (MFLOPS). In this measure the results are:

One PEM, FORTRAN	1.8 MFLOPS
One PEM, Assembly language kernel	3.6 MFLOPS
Four PEMs, FORTRAN	7.5 MFLOPS
Four PEMs, Assembly Language j line sweep	11.5 MFLOPS
Four PEMS, Assembly language iteration	17.0 MFLOPS
CDC 7600, Optimized FORTRAN	3.8 MFLOPS

The CDC 7600 rate is given for comparison. It should be noted that the entire inner loop fit within the instruction prefetch buffer of the CDC 7600. Such a buffer is not a factor in the HEP architecture. In more complex problems there would be a sharp drop in the CDC 7600 rate when the inner loop size just exceeds the prefetch buffer size. The HEP execution time for the loop will increase linearly as the loop grows.

VI. APPLICATIONS

To assess the capability of the present algorithm, it has been applied to the incompressible Navier-Stokes equations for the model problem of driven flow in a rectangular cavity. The fluid motion in the driven cavity is generated by uniform translation of the upper surface of the cavity. Two-dimensional, incompressible Navier-Stokes equations using the streamfunction ψ and vorticity ω as dependent variables have been successfully used by researchers to simulate the flow field for many applications on serial computers. Roache[8] has given a survey of the basic techniques using the

8. *P. J. Roache, "Computational Fluid Dynamics," Hermosa Publishers, Albuquerque, New Mexico, 1972.*

stream function-vorticity formulation and Tuann et. al.[9] have reviewed methods for recirculating flows.

The model problem of a driven cavity is an example of a recirculating flow or closed streamline problem and has occupied a position of considerable theoretical significance within the larger class of steady separated flows. It is known that the corner singularities present in the cavity flow cause some numerical difficulties; however, due to the local nature of these singularities the global solution is not affected significantly. Because of its geometric simplicity and the local nature of the singularities, a driven cavity flow in rectangular coordinates provides a model problem for testing new numerical schemes and algorithms and serves as a first step toward the development of an algorithm for solving the flow field about arbitrary geometrical boundary shapes. It is worth noting here that a channel flow with forward or backward facing steps can be a good model problem. However, because of the smaller number of numerical solutions available compared to the driven cavity, it was decided to investigate the driven flow in a square cavity. Previous work on this topic has been studied in detail by Burggraf.[10] Numerical schemes for the incompressible Navier-Stokes equations have been tested on driven cavity flows by several investigators.[11-16]

The governing equations for the model problem are the steady, two dimensional, laminar, incompressible Navier-Stokes equations. The velocity components are represented in terms of a stream function ψ by:

---

9. *S. Tuann and M. D. Olson, "Review of Computing Methods for Recirculating Flows," J. Comput. Phys., Vol. 29, 1978, p. 1.*

10. *O. R. Burggraf, "Analytic and Numerical Studies of the Structure of Steady Separated Flows," J. Fluid Mech., Vol. 24, 1966, p. 113.*

11. *J. D. Bozeman and C. Dalton, "Numerical Study of Viscous Flow in Cavity," J. Comput. Phys., Vol. 12, 1973, p. 348.*

12. *S. G. Rubin and J. E. Harris, "Numerical Studies of Incompressible Viscous Flow in a Driven Cavity," NASA SP-378, 1975.*

13. *M. Atias, M. Wolfshtein, and M. Israeli, "Efficiency of Navier-Stokes Solvers," AIAA Journal, Vol. 15, 1977, p. 263.*

14. *D. G. DeVahl and G. D. Mallison, "An Evaluation of Upwind and Central Differences Approximations by a Study of Recirculating Flow," J. Computer and Fluids, Vol. 4, 1976, p. 29.*

15. *K. N. Ghia, W. L. Hankey, and J. K. Hodge, "Use of Primitive Variables in the Solution of Incompressible Navier-Stokes Equations," AIAA Journal Vol. 17, 1979, p. 298.*

16. *R. Schreiber and H. B. Keller, "Driven Cavity Flows by Efficient Numerical Techniques," J. Comput. Phys., Vol. 49, No. 2, February 1983, p. 310.*

$$u = \psi_y \tag{5}$$

$$v = -\psi_x \tag{6}$$

and a vorticity is defined by:

$$\omega = u_y - v_x = \psi_{xx} + \psi_{yy} \tag{7}$$

The dimensionless momentum equation is given by the steady vorticity transport equation.

$$(u\omega)_x + (v\omega)_y = \frac{1}{Re} (\omega_{xx} + \omega_{yy}) \tag{8}$$

The Reynolds number Re is defined as Re = UL/ν, with U being the velocity of the moving wall and ν being kinematic viscosity. All velocities have been made dimensionless with respect to U; and distances, with respect to the width L of the cavity.

The boundary conditions for this problem are that normal velocity vanishes on all four walls; whereas, the tangential velocity is zero on three stationary walls and is -1 on the moving wall. The differential equations are approximated by finite-differences on a uniform mesh. Second order accurate central differences are used for all spatial derivatives. Equations (7) and (8) are solved using the parallelized point SOR relaxation procedure.

The Reynolds number R = 100 was chosen as a test case because the many numerical solutions available may be compared with the present solution. A uniform grid of 51 × 51 is considered, giving a mesh spacing of 1/50. The convergence criterion used is minimization of the residual for each difference equation in the field. The residual at each grid point is examined during the iterative process. When the maximum residual is less than the prescribed value of 10^{-5}, the solution is accepted as converged. The streamline plot of R = 100 for the present algorithm is shown in Figure 10, and the plot is in good agreement with the previously published numerical solution of Bozeman et. al.[11] for the same Reynolds number and the same mesh spacing. Since the flow is complex, a detailed comparison of the solution would be very lengthy. The quantity of the maximum stream function represents a measure of the strength of the dominant eddy inside the cavity and is a very important quantity for quantitative comparison. The maximum value of the stream function should be about 0.1 for the test case. The maximum value of the stream function reported in Reference 10 is 0.1022, in Reference 11 is 0.10316 and in Reference 17 is 0.1026. The maximum value of stream function for the present study is 0.10123.

17. *M. Nallasamy and K. K. Prasad, "On Cavity Flow at High Reynolds Number," J. Fluid Mech., Vol. 79, 1977, p. 391.*

The speedup in going from one to twenty processes executing the driven cavity code on a single PEM was 6.8. This indicates that the code was parallelized nearly as effectively as that for Laplace's equation which exhibited speedups of 7.5 and 7.2 for the optimized FORTRAN and assembly language versions, respectively. The agreement of the streamline plot with previous work and the good speedup obtained show that speedup results can be obtained for a real problem of physical interest running on a MIMD computer.

On four PEMs the speedup in going from one to 46 processes was 20.4. More than 50 processes could not be used effectively since the grid size was 50 x 50 and the program was parallelized over lines of the grid. A more complicated program or larger grid size is required to remove this limit. At a Reynolds number of 400 and with a 102 x 102 grid, a maximum speedup of 22.4 was obtained. The value of the streamfunction at the center of the upstream secondary vortex was calculated to be $-.567 \times 10^{-3}$ in comparison with a value of $-.583 \times 10^{-3}$ obtained by Nallasamy and Krishna.[17]

The major thrust of this study was to develop a relaxation algorithm in light of parallel computer architecture. In order to obtain solutions for realistic, complex fluid dynamic problems, an algorithm for the implicit finite-difference solution of the Euler and Navier-Stokes equations in general curvilinear coordinates is desired. We have also considered the unsteady vorticity transport equation

$$\omega_t + (u\omega)_x + (v\omega)_y = \frac{1}{Re}(\omega_{xx} + \omega_{yy}). \qquad (9)$$

An implicit scheme has been obtained by backward-time and central-space differencing of the derivatives. At each time step, Equation (9) is solved iteratively using the parallized SOR algorithm. The steady state solution obtained was the same as the one obtained using the steady vorticity transport equation.

A parallelized algorithm has now been developed for the implicit finite-difference solution of the time dependent incompressible Navier-Stokes equations in general curvilinear coordinates. The governing equations are in non-conservative form with the pressure and velocity as dependent variables. The computer code is currently running on the HEP and the results will be shown elsewhere.

In many applications, the implicit schemes converge faster than explicit schemes by two to three orders of magnitude. Explicit schemes are rather simple. However, the restriction on the time step imposed by stability considerations is a main disadvantage of these schemes. The high speed vector processors have made explicit schemes competitive for many applications. An explicit scheme can be easily vectorized since solving for the flow field variable at an advanced time level requires only information from the previous time step solution. Also, entire two or three dimensional grids can be considered as one long vector depending on I/O and memory capability. The use of moderate to long vectors increases computational efficiency on some vector processors. Shang, et. al.[18] and Smith, et. al.[19] have obtained considerable speedup using explicit schemes on vector processors. Similar trade-offs using explicit schemes on the parallel processor (HEP) seem possible. A parallelized algorithm for the compressible Euler equations is also currently running on the HEP. At present it looks feasible to obtain an order of magnitude speedup over similar serial code. The parallelized algorithm is more involved compared to the serial one, but the speed gain promises to reward the effort.

VII. CONCLUSIONS

The results demonstrate that it is possible to obtain a speed gain of an order of magnitude over a similar serial code using the parallelized point rowwise SOR iterative algorithm. It has been shown that the SOR method, which is not easily vectorizable, can be effectively parallelized in its original rowwise form without suffering in speed. The details of the implementation and techniques needed to make the code operate successfully on the HEP have been presented.

It was found for the present study that 17 or more parallel processes effectively utilized the computer architecture of a one PEM HEP. Some time consuming sections of the computer code were identified and effectively modified to make the code efficient. For a distinct class of problems where parallelization is more natural than vectorization, the present study suggests the possibility of both reducing the real time processing and increasing the scope of computational modeling. Since the code for a j line sweep is essentially the same as on a serial computer, the incorporation of irregularboundaries is not difficult. This is in contrast to the vector computer environment where storage rearrangement or control vector management may lead to significantly reduced performance. The specific lessons learned in this study of the development of a parallelized iterative relaxation algorithm are also valid for real codes of physical interest. This was exhibited by the good speedup obtained in the driven cavity problem where the methods developed for Laplace's equation were applied to a realistic problem.

ACKNOWLEDGEMENTS

This research was performed while the first author held an NRC - U.S. Army Resident Research Associateship at the U.S. Army Ballistic Research Laboratory. The authors wish to thank Dr. Clinton P. Frank for his help in the assembly language version of the kernel and Mr. Charles H. Breedlove for

18. *J. S. Shang, et. al., "Performance of a Vectorized Three-Dimensional Navier-Stokes Code on the CRAY-1 Computer,"* ***AIAA Journal****, Vol. 8, No. 9, September 1980.*

19. *R. E. Smith, et. al., "The Solution of the Three-Dimensional Viscous Compressible Navier-Stokes Equations on a Vector Computer,"* ***Advances in Computer Methods for Partial Differential Equations****, IMACS, 1979.*

his assistance in running and debugging the program. Computer support for this work has been provided by the U.S. Army Ballistic Research Laboratory. Thanks also to Mr. Robert Reschly, Jr. for running the CDC 7600 version of the code.

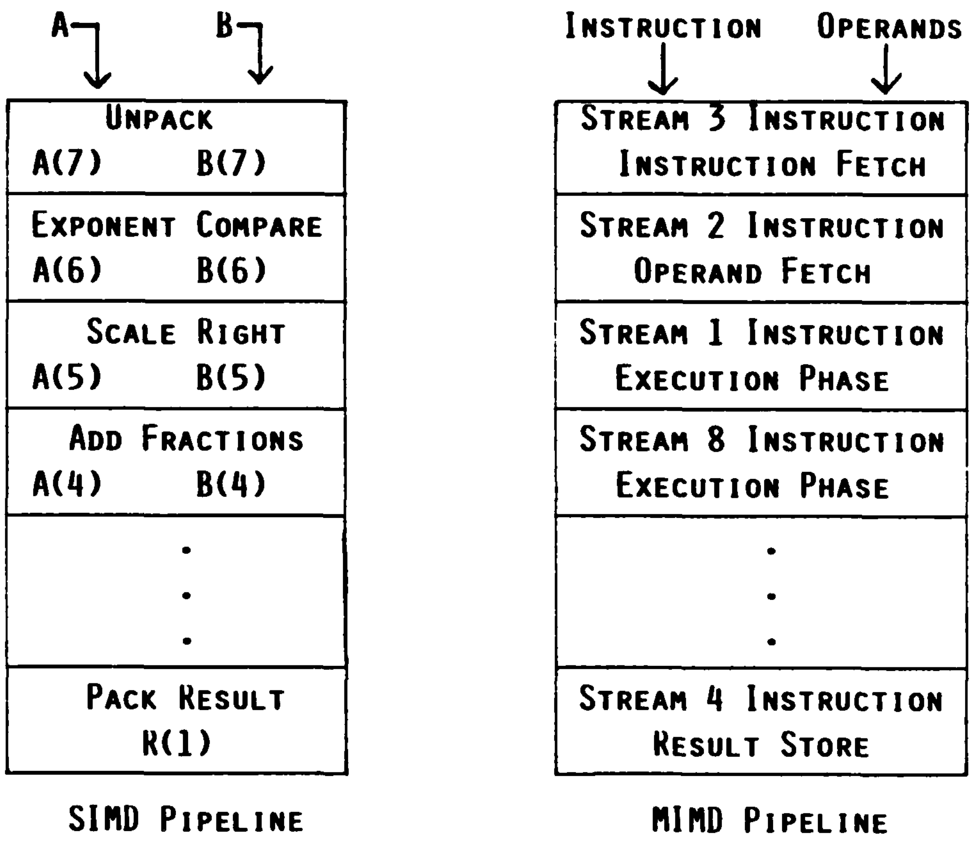


Figure 1. SIMD Versus MIMD Pipelining

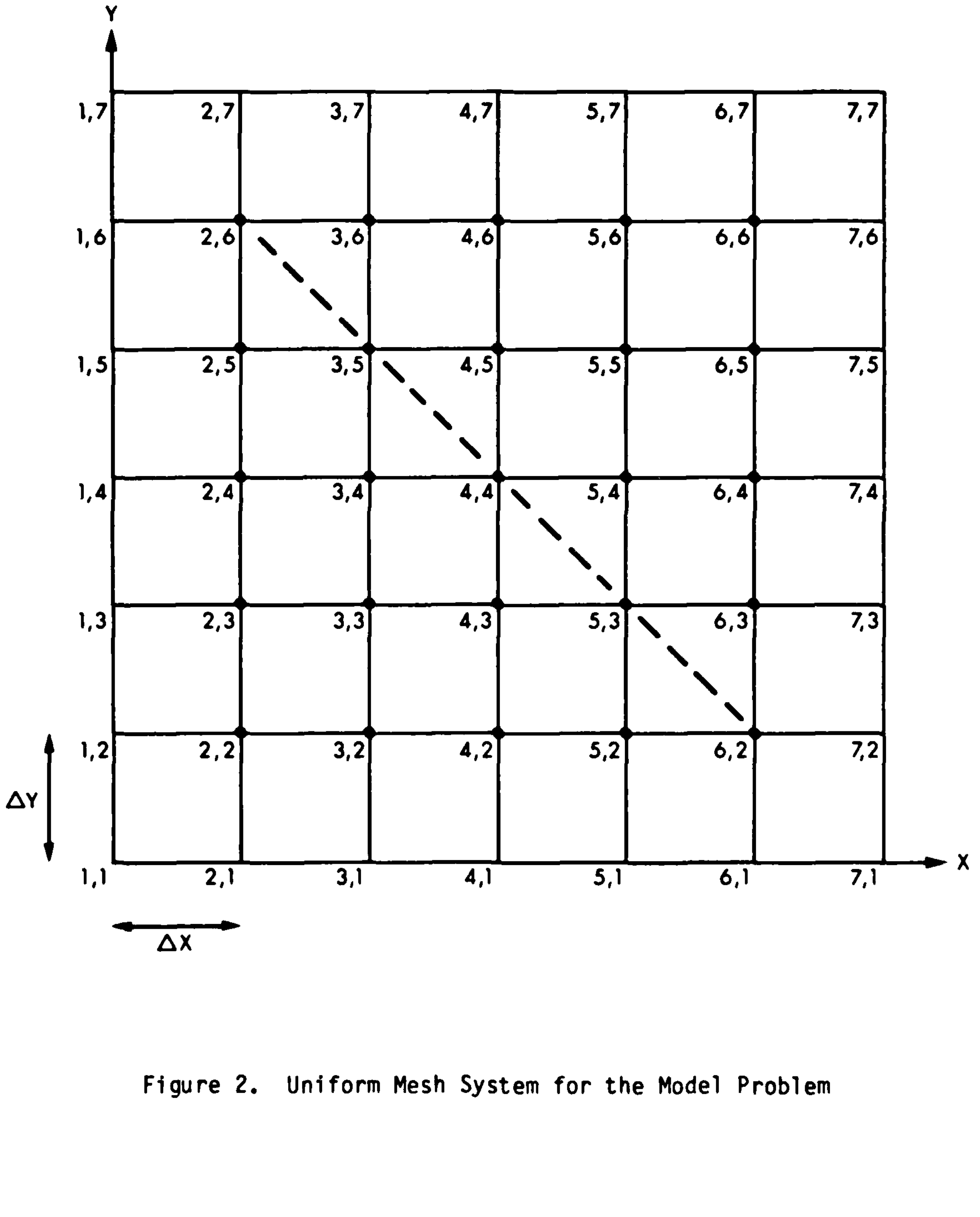


Figure 2. Uniform Mesh System for the Model Problem

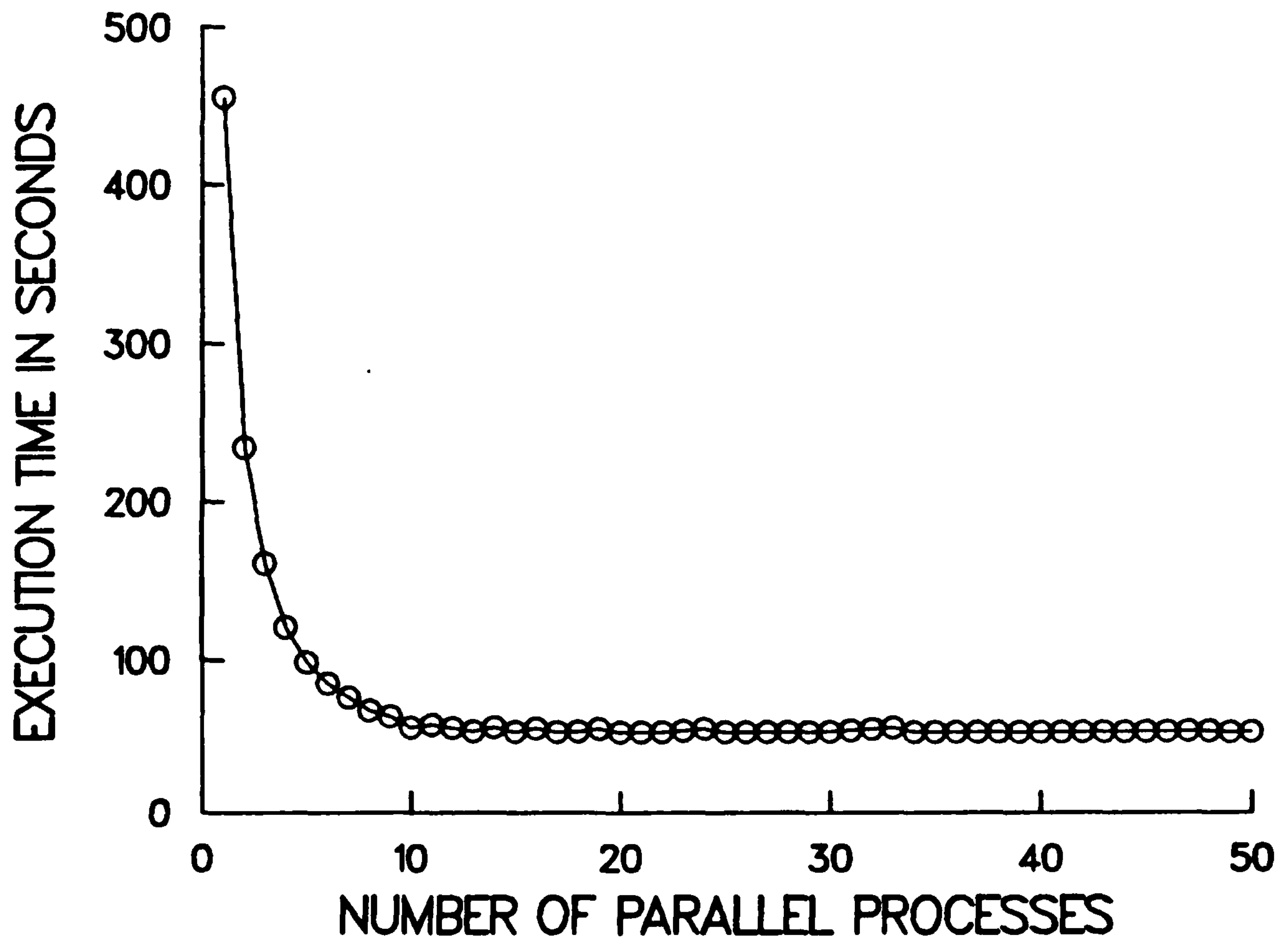


Figure 3. Speed of First FORTRAN Version

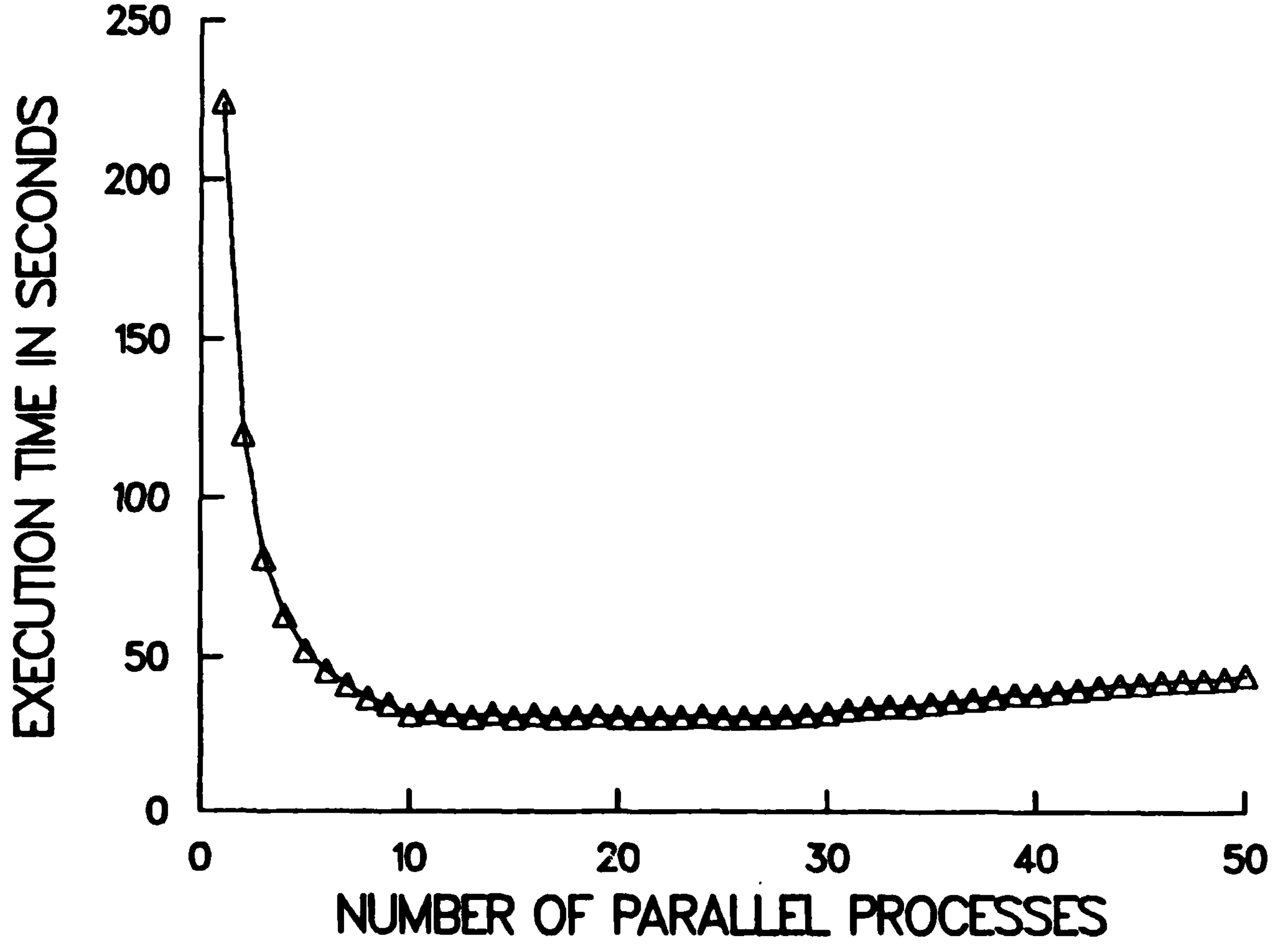


Figure 4. Speed of Optimized FORTRAN Version

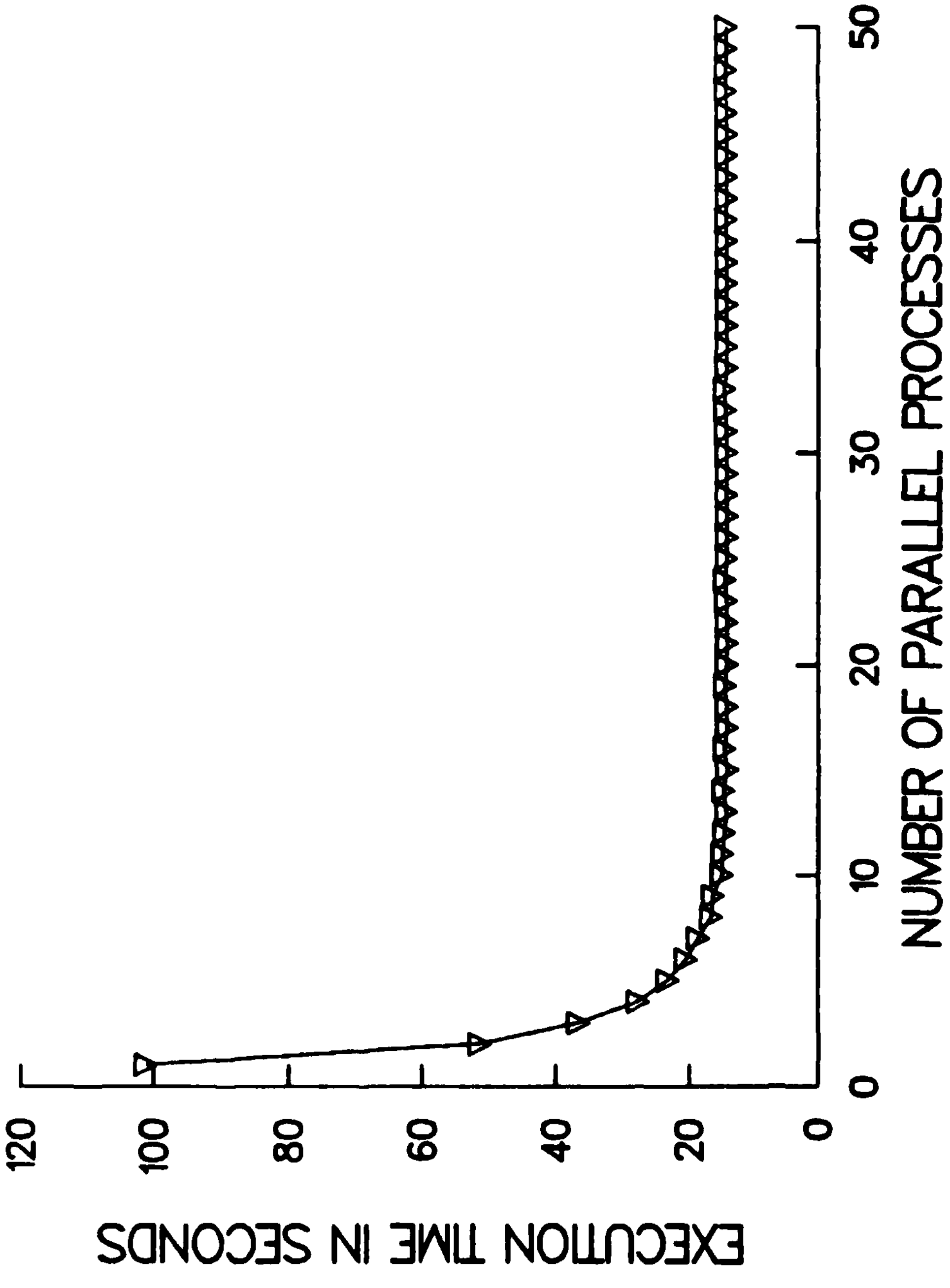


Figure 5. Speed of Version with Assembly Language Kernel

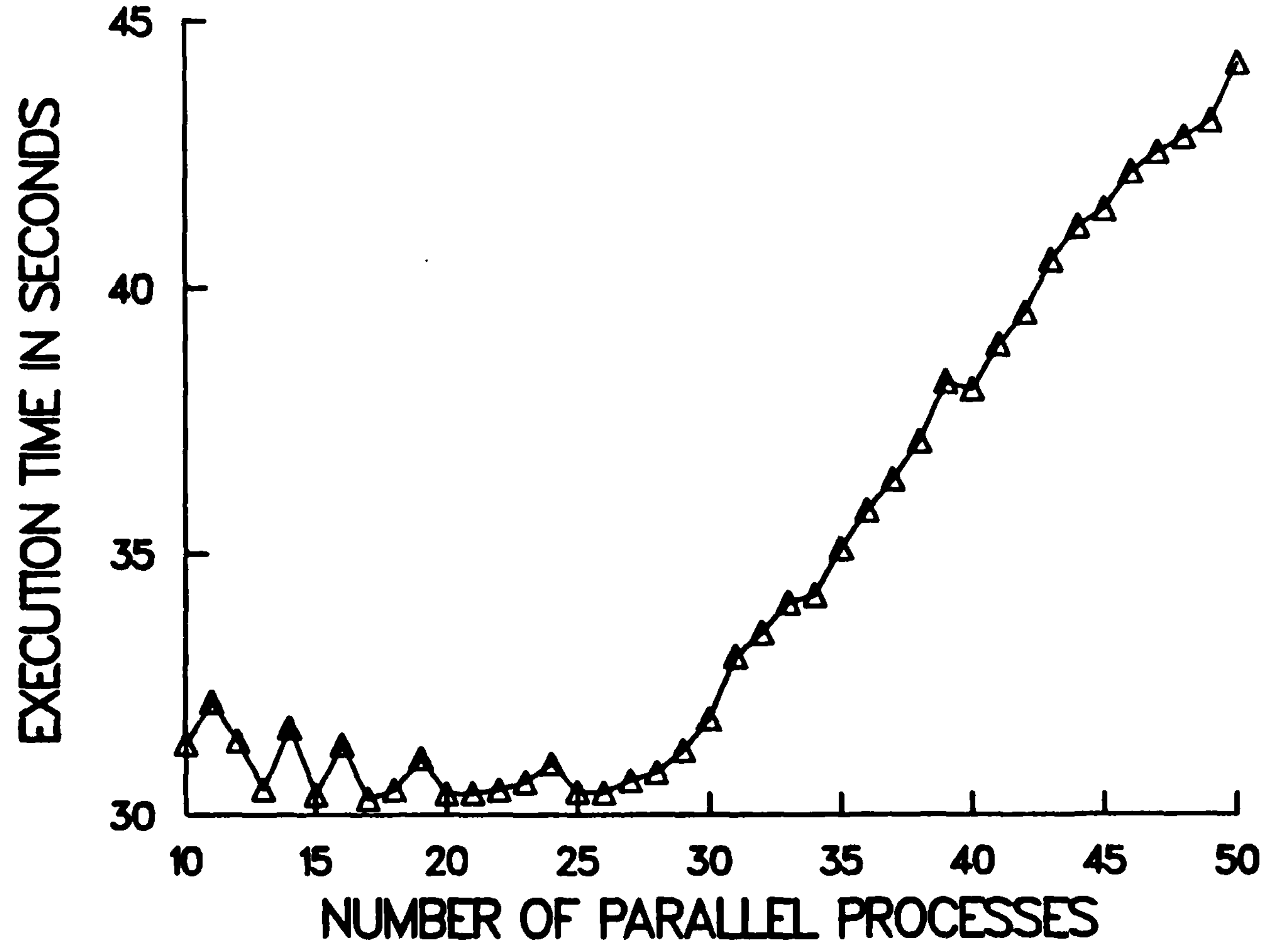


Figure 6. Effects of the Critical Section for Many Processes

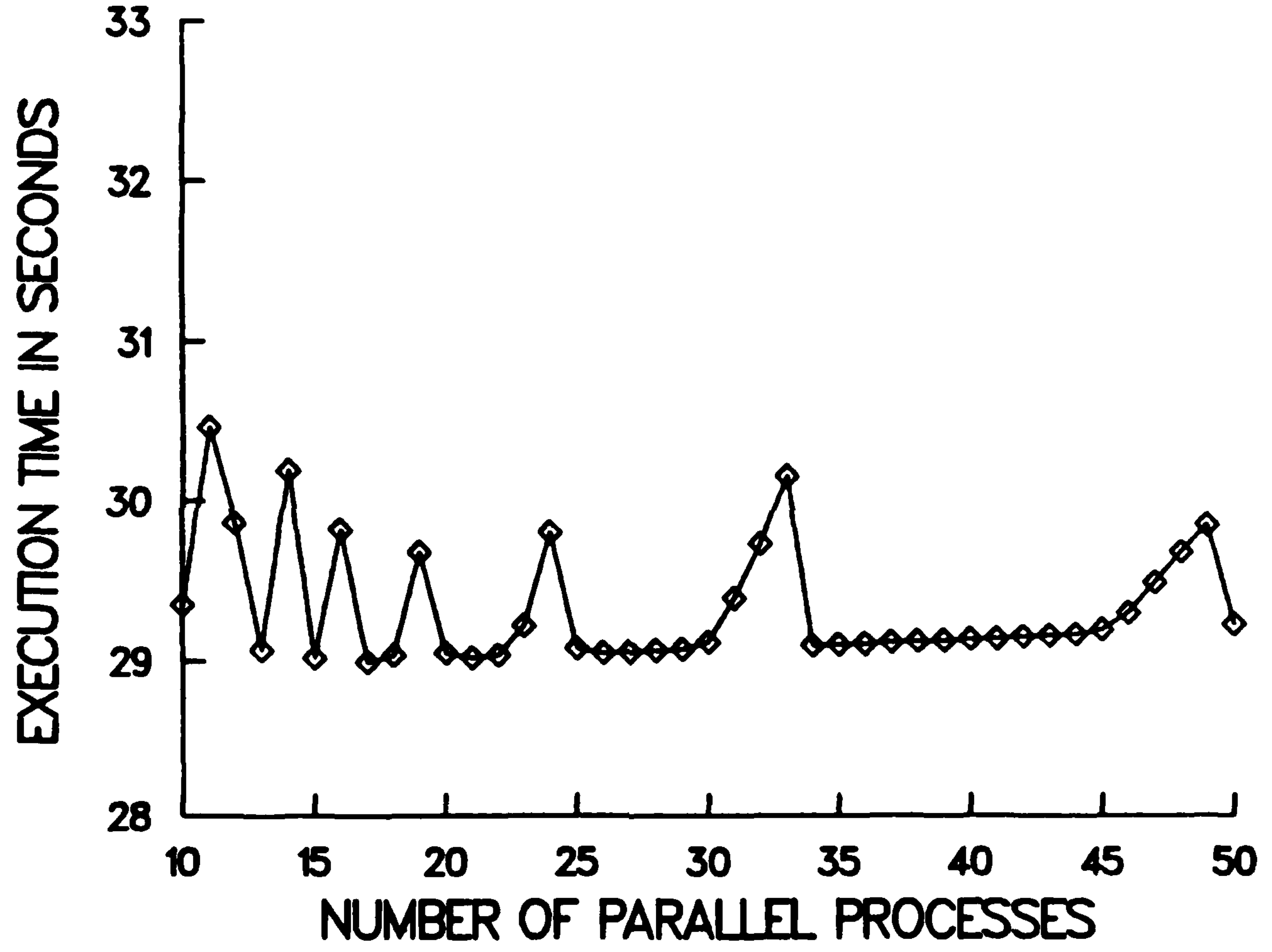


Figure 7. Critical Section Removed from the Inner Loop

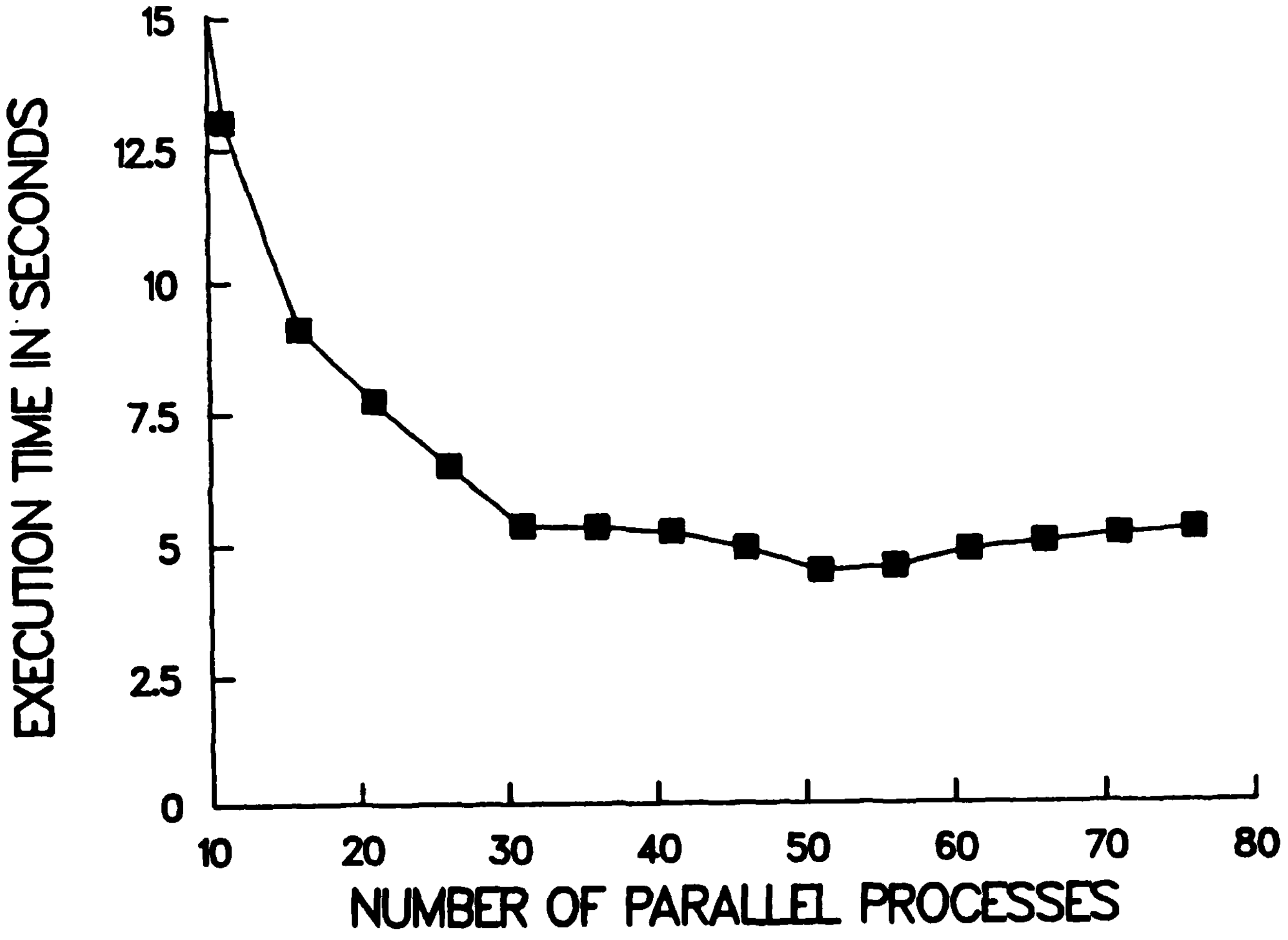


Figure 8. Speed of Four PEM Version with Assembly Language j Line Sweep

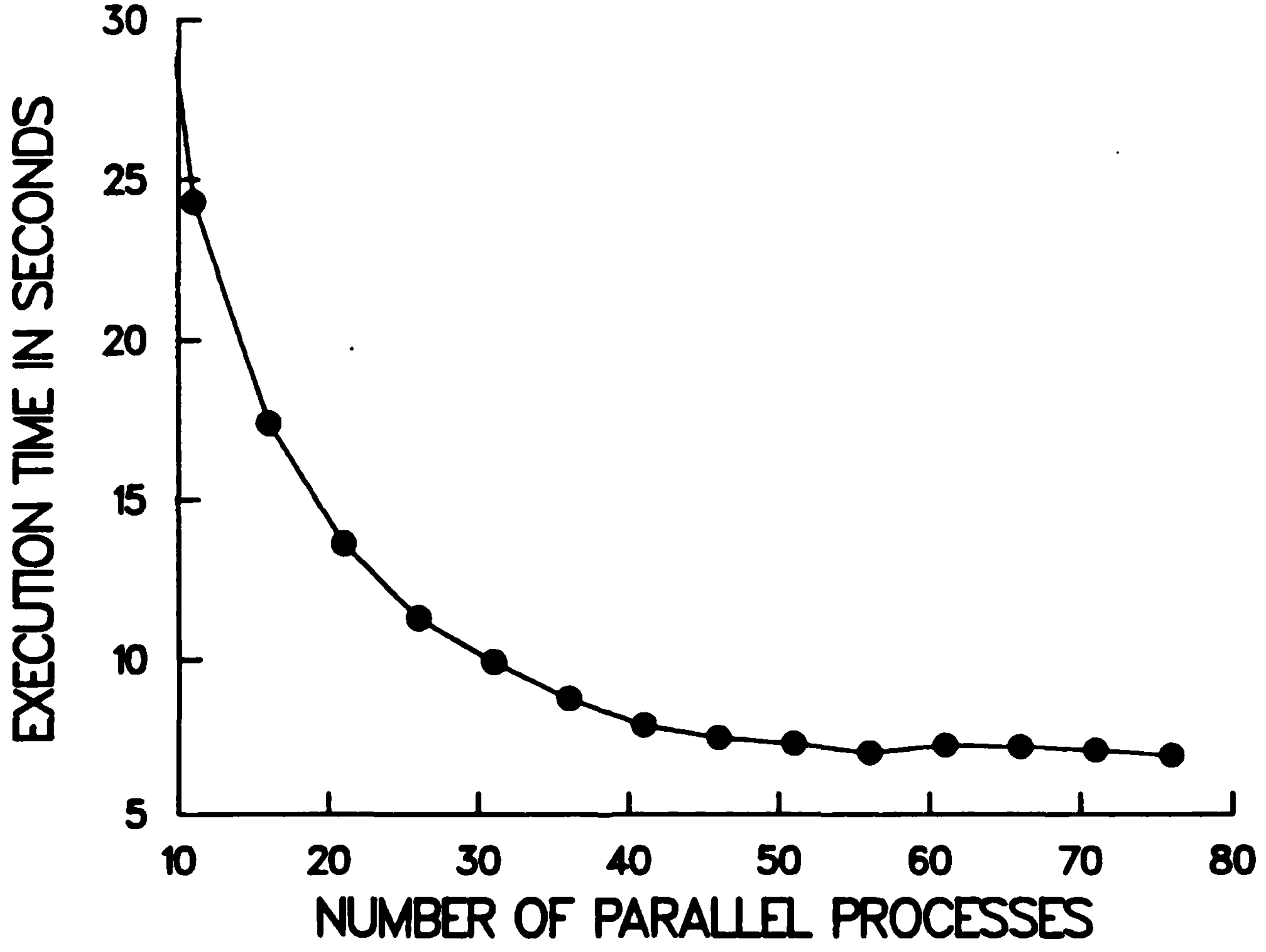


Figure 9. Speed for Four PEM FORTRAN Version

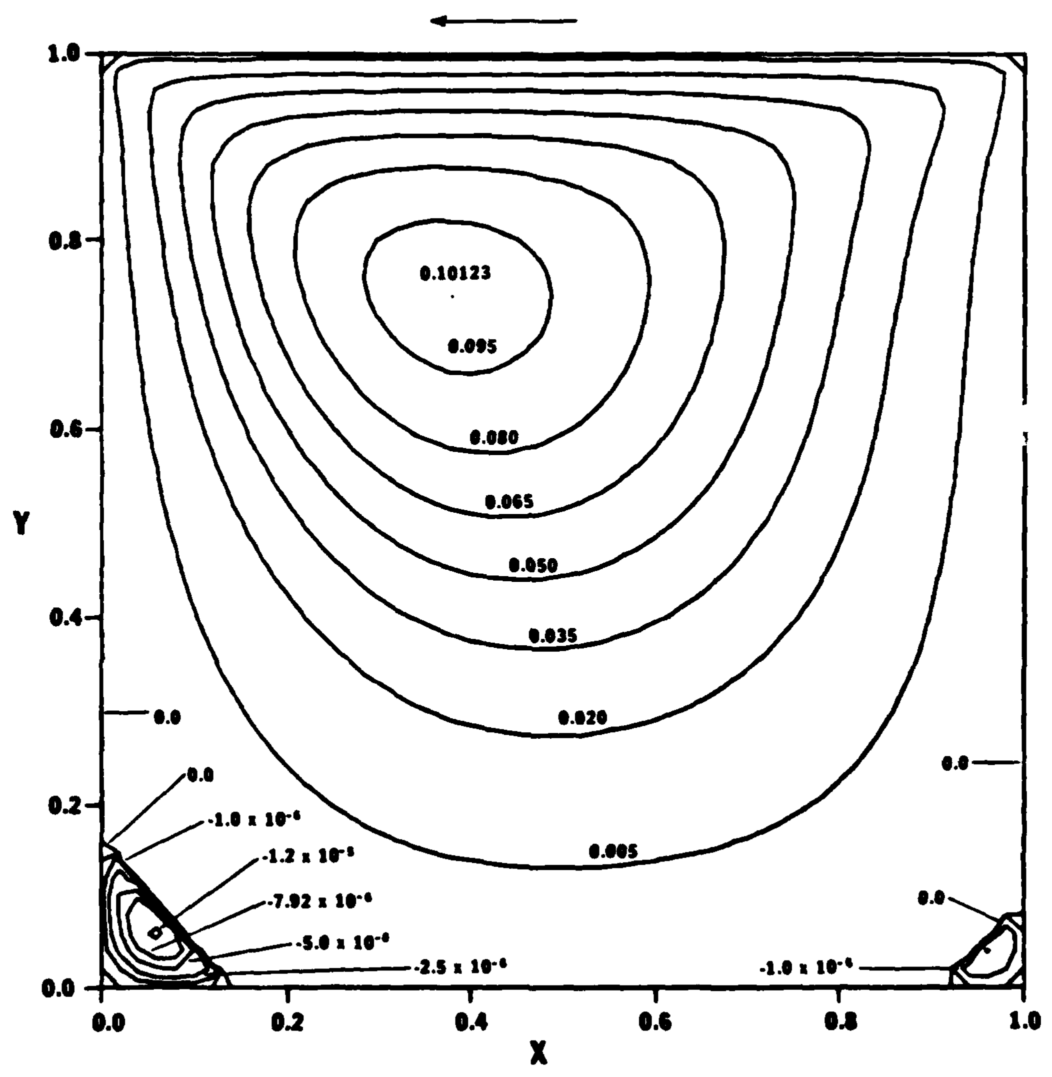


Figure 10. Streamline Patterns and Stream Function Values for Flow in a Square Cavity, Re = 100

REFERENCES

1. R. D. Levin, "Supercomputers," Scientific American, January 1982, p. 118.

2. H. F. Jordan, "Experience with Pipelined Multiple Instruction Streams," Proc. IEEE, Vol. 72, No. 1, January 1984, pp. 113-213.

3. HEP FORTRAN 77 USER'S GUIDE, Reference Manual, Denelcor Inc., Denver, Colorado, February 1982.

4. R. S. Varga, Matrix Iterative Analysis, Prentice-Hall Inc., New York, New Jersey, 1962

5. L. M. Adams and J. M. Ortega, "A Multi-Color SOR Method for Parallel Computation," Proc. 1982 Intl. Conf. on Parallel Processing, Bellaire, MI, August 1982, pp. 53-58.

6. P. F. Bradley, D. L. Dwoyer and J. C. South, "Vectorized Schemes for Conical Flow Using the Artificial Density Method," AIAA Paper Number 84-0162, January 1984.

7. N. Patel and J. F. Thompson, "A Vectorized Solution for Incompressible Flow," AIAA Paper No. 84-1534, June 1984.

8. P. J. Roache, "Computational Fluid Dynamics," Hermosa Publishers, Albuquerque, New Mexico, 1972.

9. S. Tuann and M. D. Olson, "Review of Computing Methods for Recirculating Flows," J. Comput. Phys., Vol. 29, 1978, p. 1.

10. O. R. Burggraf, "Analytic and Numerical Studies of the Structure of Steady Separated Flows," J. Fluid Mech., Vol. 24, 1966, p. 113.

11. J. D. Bozeman and C. Dalton, "Numerical Study of Viscous Flow in Cavity," J. Comput. Phys., Vol. 12, 1973, p. 348.

12. S. G. Rubin and J. E. Harris, "Numerical Studies of Incompressible Viscous Flow in a Driven Cavity," NASA SP-378, 1975.

13. M. Atias, M. Wolfshtein, and M. Israeli, "Efficiency of Navier-Stokes Solvers," AIAA Journal, Vol. 15, 1977, p. 263.

14. D. G. DeVahl and G. D. Mallison, "An Evaluation of Upwind and Central Differences Approximations by a Study of Recirculating Flow," J. Computer and Fluids, Vol. 4, 1976, p. 29.

15. K. N. Ghia, W. L. Hankey, and J. K. Hodge, "Use of Primitive Variables in the Solution of Incompressible Navier-Stokes Equations," AIAA Journal, Vol. 17, 1979, p. 298.

16. R. Schreiber and H. B. Keller, "Driven Cavity Flows by Efficient Numerical Techniques," J. Comput. Phys., Vol. 49, No. 2, February 1983, p. 310.

REFERENCES (continued)

17. M. Nallasamy and K. K. Prasad, "On Cavity Flow at High Reynolds Number," J. Fluid Mech., Vol. 79, 1977, p. 391.

18. J. S. Shang, et. al., "Performance of a Vectorized Three-Dimensional Navier-Stokes Code on the CRAY-1 Computer," AIAA Journal, Vol. 8, No. 9, September 1980.

19. R. E. Smith, et. al., "The Solution of the Three-Dimensional Viscous Compressible Navier-Stokes Equations on a Vector Computer," Advances in Computer Methods for Partial Differential Equations, IMACS, 1979.

DISTRIBUTION LIST

No. of Copies	Organization
12	Administrator Defense Technical Info [illegible]enter ATTN: DTIC-DDA Cameron Station Alexandria, VA 22314
1	HQDA DAMA-ART-M Washington, DC 20310
1	Commander US Army Materiel Command ATTN: AMCDRA-ST 5001 Eisenhower Avenue Alexandria, VA 22333
8	Commander Armament R&D Center US Army AMCCOM ATTN: SMCAR-TDC SMCAR-TSS SMCAR-LCA-F Mr. D. Mertz Mr. E. Falkowski Mr. A. Loeb Mr. R. Kline Mr. S. Kahn Mr. H. Hudgins Dover, NJ 07801
1	Commander US Army Armament, Munitions and Chemical Command ATTN: AMSMC-LEP-L Rock Island, IL 61299
1	Director Benet Weapons Laboratory Armament R&D Center US Army AMCCOM ATTN: SMCAR-LCB-TL Watervliet, NY 12189
1	Commander US Army Aviation Research and Development Command ATTN: AMSAV-E 4300 Goodfellow Blvd St. Louis, MO 63120
1	Director US Army Air Mobility Research and Development Laboratory Ames Research Center Moffett Field, CA 94035
1	Commander US Army Communications Research and Development Command ATTN: AMSEL-ATDD Fort Monmouth, NJ 07703
1	Commander US Army Electronics Research and Development Command Technical Support Activity ATTN: AMDSD-L Fort Monmouth, NJ 07703
2	Commander US Army Missile Command ATTN: AMSMI-RDK (Dr. W. Walker) (Mr. R. Deep) Redstone Arsenal, AL 35898
1	Commander US Army Missile Command ATTN: AMSMI-YDL Redstone Arsenal, AL 35898
1	Commander US Army Tank Automotive Command ATTN: AMSTA-TSL Warren, MI 48090
1	Director US Army TRADOC Systems Analysis Activity ATTN: ATAA-SL White Sands Missile Range, NM 88002
1	Commander US Army Research Office P. O. Box 12211 Research Triangle Park, NC 27709-2211

DISTRIBUTION LIST

No. of Copies	Organization
1	Commander US Naval Air Systems Command ATTN: AIR-604 Washington, D. C. 20360
2	Commander David W. Taylor Naval Ship Research and Development Center ATTN: Dr. S. de los Santos Mr. Stanley Gottlieb Bethesda, Maryland 20084
1	Commander US Naval Surface Weapons Center ATTN: Code DK20 Dahlgren, VA 22448
1	Commander US Naval Surface Weapons Center ATTN: Code R44 Dr. T. Zien Silver Spring, MD 20910
1	Commander US Naval Weapons Center ATTN: Code 3433, Tech Lib China Lake, CA 93555
1	Commander US Army Development & Employment Agency ATTN: MODE-TED-SAB Fort Lewis, WA 98433
1	Commandant US Army Infantry School ATTN: ATSH-CD-CSO-OR Fort Benning, GA 31905
1	AFWL/SUL Kirtland AFB, NM 87117
1	ACUREX Corporation/Aerotherm Div ATTN: Mr. W. S. Kobayashi 555 Clyde Avenue P.O. Box 7555 Mountain View, CA 94039
1	University of Colorado Electrical & Computer Engr Dept ATTN: Harry A. Jordan Boulder, CO 80309
1	Commander US Army Missile Command ATTN: AMSMI-R Redstone Arsenal, AL 35898
3	Director Argonne National Laboratory ATTN: Dr. E. L. Lusk Dr. R. Overbeek Dr. J. Dongarra 9700 South Cass Avenue Argonne, IL 60439
1	Denelcor, Inc. ATTN: Dr. B. J. Smith 17000 East Ohio Place Aurora, CO 80012
1	Director Goodyear Aerospace Corporation ATTN: D/470 (Dr. K. E. Batcher) 1210 Massillon Road Akron, OH 44315
1	Director IBM Thomas J. Watson Research Center ATTN: Dr. F. Ris P.O. Box 218 Yorktown Heights, NY 10598
2	Director Lawrence Livermore National Lab. ATTN: L360 (Dr. G. Rodrique) (Dr. G.A. Michael) P.O.Box 808 Livermore, CA 94550
1	Director Lawrence Livermore National Lab. ATTN: L-71 (Dr. T. S. Axelrod) P.O. Box 808 Livermore, CA 94550
3	Director Los Alamos National Laboratory ATTN: MS B260 (Dr. B.L. Buzbee) MS B265 (Dr. J.W. Moore) (Dr. O.M. Lubeck) P.O. Box 1663 Los Alamos, NM 87544

DISTRIBUTION LIST

No. of Copies	Organization
1	Director NASA Ames Research Center NAS Projects Office ATTN: MS 202-1, (Dr. F.R. Bailey) Moffett Field, CA 94035
1	Director NASA Langley Research Center ATTN: F. C. Thames Hampton, VA 23365
1	Director Oregon Graduate Center Department of Computer Science and Engineering ATTN: Dr. R. Babb 19600 NW Walker Road Beaverton, OR 97006
2	Director Sandia National Laboratory ATTN: Technical Staff, Dr. W.L. Oberkampf Aeroballistics Division 5631, H.R. Vaughn Albuquerque, NM 87184
1	Massachusetts Institute of Technology ATTN: Tech Library 77 Massachusetts Avenue Cambridge, MA 02139
1	Virginia Polytechnic Institute & State University ATTN: Dr. Clark H. Lewis Department of Aerospace & Ocean Engineering Blacksburg, VA 24061
1	University of Delaware Mechanical and Aerospace Engineering Department ATTN: Dr. J. E. Danberg Newark, DE 19711
1	Mississippi State University Department of Aerospace Engr ATTN: Dr. J. F. Thompson Mississippi State, MS 39762
1	Mississippi State University Department of Applied Mathematics ATTN: Dr. C. W. Mastin Mississippi State, MS 39762
1	New York University Courant Institute Mathematical Sciences ATTN: Dr. M. H. Kalos 251 Mercer Street New York, NY 10012
1	Purdue University Department of Computer Sciences ATTN: Dr. D. B. Gannon West Lafayette, IN 47906
1	Rutgers University Department of Aerospace Engr ATTN: Dr. D. G. Briggs Piscataway, NJ 08854
1	University of Texas at Austin Department of Computer Sciences ATTN: Dr. J. C. Browne Austin, TX 78712
1	Virginia Polytechnique Institute and State University Department of Aerospace and Ocean Engineering ATTN: Dr. C. Lewis Blacksburg, VA 24061

<u>Aberdeen Proving Ground</u>

Dir, USAMSAA
ATTN: AMXSY-D
AMXSY-MP, H. Cohen

Cdr, USATECOM
ATTN: AMSTE-TO-F

Cdr, CRDC, AMCCOM
ATTN: SMCCR-RSP-A
SMCCR-MU
SMCCR-SPS-IL

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. BRL Report Number ______________________ Date of Report __________

2. Date Report Received ______________________________________

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) __________

4. How specifically, is the report being used? (Information source, design data, procedure, source of ideas, etc.) __________

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided or efficiencies achieved, etc? If so, please elaborate. __________

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.)

CURRENT ADDRESS

____
Name

____
Organization

____
Address

____
City, State, Zip

7. If indicating a Change of Address or Address Correction, please provide the New or Correct Address in Block 6 above and the Old or Incorrect address below.

OLD ADDRESS

____
Name

____
Organization

____
Address

____
City, State, Zip

(Remove this sheet along the perforation, fold as indicated, staple or tape closed, and mail.)

— — — — — — — — — — — FOLD HERE — — — — — — — — — — — — —

Director
US Army Ballistic Research Laboratory
ATTN: AMXBR-OD-ST
Aberdeen Proving Ground, MD 21005-5066

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE, $300

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 12062 WASHINGTON,DC

POSTAGE WILL BE PAID BY DEPARTMENT OF THE ARMY

Director
US Army Ballistic Research Laboratory
ATTN: AMXBR-OD-ST
Aberdeen Proving Ground, MD 21005-9989

— — — — — — — — — — — FOLD HERE — — — — — — — — — — — — —